Common Areas At The Heart

George Dinwiddie
iDIA Computing, LLC
gdinwiddie@idiacomputing.com

Abstract

In Christopher Alexander's pattern of this name, he says, "No social group—whether a family, a work group, or a school group—can survive without constant informal contact among its members." [1] For a software development team, the practice of sitting together goes a long way in providing this informal contact and, ultimately, making it a team instead of a collection of individuals assigned to a project. It is easily the best first step toward agility.

1. Introduction

As has happened several times in my career, I was between jobs due to a lay-off at my previous employer. Uncharacteristically, I had been between jobs for several months—the job market was tight. When I found an opportunity, it came with a warning from a colleague who was working there: time was short, pressure was high, and things weren't going very well. The project was a re-implementation of an e-commerce web site, switching from Perl to Java. The requirements were simple: do everything the old site does, without the bugs, and adding a bunch of new features that were still being dreamed up.

Besides needing the income, I was optimistic that I could make a difference. I had become "test infected" and found my software skills had blossomed like never before. I had become active with the Washington DC XP group, helped found the Maryland XP group, and had significant understanding of the practices of Agility. I took the job.

2. The initial situation

Of course, I entered this situation with little or no credibility. I was a newcomer, and a contractor, at that. I started on the same day as five other contractors. We were given a high-level introduction to the project and installed in a room with computers on tables. Ah, a team room? Nope, we were given individual task-level assignments. "Create a screen with these fields." "Build a module that does this." Not much about how the assignments fit into the whole.

I was fortunate in that my first assignment was relatively self-contained and suitable for a one-person mini-project. Test Driven Development allowed me to deliver high quality code in minimal time—that helped my credibility. As the code came into shape, I asked about the API that would be appropriate for the client code. "Oh, that hasn't been done, yet." My module was a disconnected bit of code floating in the weightlessness of the void.

I've never been comfortable with the concept that software can be made in lots of little pieces and then connected together. Oh, it can be done that way, but the resulting code soup never seems to have a pleasing flavor. Even when it works, there's a lack of conceptual integrity.

Looking for the bigger picture, the organizing viewpoint, I got to know the other developers, and asked them questions about the design. That helped my credibility, too. In this shop it seemed unusual for contractors to take an interest in the business. The answers, though, were not always in agreement. Asking two people together generally resulted in a discussion about the advantages of this architecture or that architecture. There didn't seem to be a consensus of understanding.

3. A chance to change

The combination of uncertain direction and accelerating change in requirements finally came to a head and big steps were taken. Development virtually ceased while the business made some decisions about the requirements. Large changes were made in the development team structure—what had been one team became four, presupposing some architectural decisions along political boundaries.

To be honest, I wasn't sanguine about some of these changes, but the pause in the furious pace of development allowed time for some discussion about how things were being done. I had, by this time, made a good enough impression to have some of my ideas heard and considered. Not everything that I suggested went somewhere. A day's worth of writing story cards and estimating them was thrown out in favor of a Microsoft Project plan that listed tasks I couldn't always recognize (like the way that KFC cuts chickens so that there are no back pieces). A suggestion of pair programming was soundly rejected as wasteful. But one idea took root—that of moving the team out of scattered cubicles and into a common room.

This simple act is, in many circumstances, very difficult to accomplish. The first difficulty is often that of finding a space. The cost of office space often means that all available space is used, with none left for a team to use. Converting cubicles to an open area for the team generally requires the cooperation with a facilities group, who may be more interested in their efficiency of operation than the production of business value by software development. As Jerry Weinberg has observed, "Who is the most important process person? The one who arranges the furniture." [2]

Sometimes a team can take over a conference room. In this instance, there was a room that had obviously been a server or network room at a previous time. More recently it had been furnished with desk surfaces and used as workspace for contractors. This furniture wasn't ideal, as it had the intent to separate the individual workspaces, but it did so less effectively than cubicle walls. The room wasn't considered a desirable working space. The one window at the end of the room looked out on a passageway, not the outside world. The rooms of cubicles, by contrast, had large windows letting in natural light.

4. Reluctance of the team

I pitched the idea to the team manager on the basis of increased communication, keeping the team collectively working to the same design. The manager, in turn, had the credibility with other team members that they willingly, if reluctantly, agreed.

One developer, who had an outside corner cubicle, with exterior windows on both sides, sadly remarked, "I'll never get that cube back." I spoke with him recently to get his viewpoint and he mentioned a couple of deeper concerns. Not only was that workspace desirable in terms of being illuminated with natural light and having a view, but it was a personal workspace. People have a strong need for spaces that they control and call their own. Moving into a shared space gives up that complete control and requires negotiation and cooperation to exert influence.

Another concern was that the noise of a shared room would be distracting and disruptive. Software development is often thought of as a solo endeavor primarily, I think, because of the concentration required to get the ideas in our heads expressed in code on the machine. Many people have written about the concept of "flow" and the high cost of interruptions, even short ones, to that concentration. DeMarco and Lister describe some figures from the study that preceded the construction of IBM's Santa Teresa facility. [3] These figures suggest that software developers (in IBM's culture of the mid-1970s, at least) work with others, generating noise, 70% of the time and work alone, sensitive to interruption, the remaining 30%.

The developer, however, agreed that it was the best thing for the project. This willingness is very important, for I have seen the team room approach fail terribly when forced on developers against their will. I observed, at a different company, a project that had moved into a "war room" at the insistence of the project leader, but without the willingness of the team. The project risk assessment addressed many identified potential risks as being avoided or mediated "because the development will take place in a war room" without addressing how the war room solved these issues. In fact, the developers who moved into this space rearranged the furniture, dividing workspaces will tall bookcases and arranging desks such that none of them could see other developers while working alone at their computers. Seeking the privacy and familiarity of the offices they'd reluctantly left, they recreated them as best as possible—negating the shared space intended by a common work area.

5. Team manager involvement

The team manager also moved into the room, acting as a Customer proxy by resolving questions about the details of the requirements, and protecting the team from interference by sitting closest to the door. One developer described this as doing a "Doberman impression" because he'd abruptly interrupt his work to halt and interrogate anyone entering the room. Many times he could provide the needed information, himself, without interrupting the development effort. Other times he could determine that the stated need indicated that some deeper issue needed exploring, and would leave with the interloper to resolve the understanding of that issue. Of course, when there was a genuine need to talk about the software and how the modules worked

together, he readily allowed immediate access for discussion.

He also provided an excellent example, arriving early but not leaving until the last programmer left. As stated by Wain's Fifth Conclusion, “Nothing motivates a man more than to see his boss put in an honest day's work.” [4] The company expectations were not for maintaining a sustainable pace. Instead, the larger organization seemed to believe that the harder you push developers, the sooner the software will be done. In this atmosphere, sometimes long days were required. Late changes or additions to the requirements for a release would be mandated without modification of the due date. On one occasion, I remember working until midnight to finish revising a feature that had been changed in definition the day before an internal release to QA. He was right there with me, working on paperwork and plans and such, and we left the building together. And while I delayed my arrival an hour the next day, to 10:00, he was there at his normal 8:00. It's amazing how much this willingness to share the pain of the team contributed to the cohesiveness of the team.

6. Working in the team room

Working together in the team room changed the way the team worked in a number of ways.

6.1. Feature by feature, page by page

Rather than working at individual tasks in isolation, the team tended to work on tasks related to a single feature. The user flow of the feature would be documented on large newsprint pages posted on the wall, describing the contents of the pages, the choices of the end user, and the navigation results of each of those choices. The team manager kept these up-to-date with changes from the UI design team, which was in a remote site from the development team.

On a typical feature, some of the developers would be implementing the JSP pages, while others would be implementing the back-end functionality to support those pages. While the GUI/logic assignments tended to be rather static due to the skill sets of the developers, there was frequent communication between the two to coordinate the efforts. The GUI developers learned to speak up when they needed some information in a different manner for display. This contrasts with the tendency to add calculations in the display code when each developer was assigned individual tasks. The business logic developers learned, likewise, to have a discussion with the page designer before deciding on an API to support that page.

As features were implemented, they were marked on the papers covering the wall. This gave a clear indication of what had been done, and what was left to do. Completed pages were shown to the remote UI and QA teams, allowing early feedback about any issues with them.

6.2. Design discussions

Design discussions were frequent occurrences. Many times they began with one developer asking another for opinions on the design they had in mind for a feature. The other developer might offer suggestions, or discuss how the design fit with the existing code. Often this would precipitate some sketching of alternatives on the whiteboard. These whiteboard sessions invariably drew others into the conversation. The extra participation honed the design of the feature under discussion, and sometimes improved the overall system design significantly. There were times when design proposals for a feature conflicted with the needs of another feature. Collaboratively, the team always came up with a way to separate the conflicting needs and satisfy them both.

6.3. Learning techniques

Nobody knows everything. In the common cubicle environment of traditional software development organizations, there is precious little opportunity to observe what others are doing and thereby increase the breadth of your skills. In the team room, learning opportunities abound. A team member told me that he learned new design techniques by eavesdropping on the design sessions of others.

Technology knowledge was also spread between team members. For example, I learned how to deal with quirks of certain JSP tags in the Struts Tag Library by asking nearby GUI developers for help. As we had a mix of native languages, often the quickest way to share knowledge was to demonstrate it. Ad-hoc pair programming would spring up for a few moments as the technique was demonstrated. Indeed, where I'd been primarily a J2EE back-end developer, I learned enough to comfortably develop JSP pages as well.

Tool proficiency was another area that benefited by working in close proximity. Everyone described new techniques and shortcuts they'd discovered in Eclipse. Soon everyone else was relying on the same techniques to make coding more efficient.

Business knowledge was also spread by the osmosis of the team room. The metaphor of a shopping cart on an e-commerce web site seems like a simple one. The

products for sale were services rather than physical, shippable items. There were interrelationships between the products that generated complicated business rules. “You can only buy this product if you've previously, or concurrently, bought that one.” “For each of that product you own, you can purchase, at most, one of this type of product.” Some of the interactions between these business rules were subtle, or even surprising.

6.4. Alignment of understanding

As knowledge transferred from person to person, so did the understanding. Knowledge of both the problem to be solved and the way in which it was being solved was common throughout the team. I've always said that any person on a project should be able to sketch a high-level view of the system being developed. This team was the first time I've actually experienced that. Most of my experiences have been that people knew only their small part of the whole, and not the big picture. In other cases, a system architect (or design document) had the only complete picture and many of the developers deferred to that source, acquiring little understanding at all.

This shared understanding of the system contributed greatly to the conceptual integrity of the system. Fred Brooks emphasizes that “conceptual integrity of the product not only makes it easier to use, it also makes it easier to build and less subject to bugs.” [5] While Brooks recommends a single architect to provide this unity, he does ask, “How does one ensure that every trifling detail of an architectural specification gets communicated to the implementor, properly understood by him, and accurately incorporated into the product?” [6] The team experience described here offers one path to assurance.

It should also be noted that the design produced by collaboration had more conceptual integrity than those on sister teams that were produced principally by a single person. The lesson derived from this is that disunity is not generated from a multitude of people working on the design, but from people working on the design in a piecemeal fashion. Certainly it's as possible for a single individual to work in a piecemeal fashion as it is for a team to work in a unified way.

6.5. Sense of 'teamness'

The simple act of working together produced some marvelous results. Whiteboard discussions of strategy and design involved the whole team. The pageflow design covering the walls kept everyone informed on what was done, and what needed to be done. New ideas, and erroneous assumptions, would be overheard within the room and supported or corrected as appropriate. Ad-hoc pair programming sprouted here and there like wildflowers after a spring rain.

Not all of the teamwork was businesslike, however. The team manager and members took to bringing in snack in sufficient quantities to put out for the team to share. A new team member joining the group from another project brought the putter and golf ball he kept in his cubicle. Soon there were two putters, several golf balls, and a practice hole and the whole team (mostly not golf players) could be found taking a few practice strokes when thinking about or discussing a puzzling issue. Even the CTO would practice a few shots when he came in the room. Jokes were common, and some of the funniest statements were memorialized on a list on the wall.

The team jelled, despite ongoing perturbations in team composition and project definition. In fact, the biggest negative was that other teams complained that we had too much fun and they felt left out. We offered our snacks to them and made a concerted effort to eliminate jokes that were at the expense of other teams.

Productivity improved enough, and the feature-by-feature approach made the progress visible enough, that the team managed a few outings in spite of the continued high schedule pressure. On one occasion, a team member invited everyone to his house for some home-cooked Indian food. I still yearn for more of his home-made yoghurt. Another entire day was spent sailing to a restaurant five hours away for lunch, and then back. I'm sure the team manager caught some heat for “wasting time” like this, but it was time well spent.

Despite the continued less-than-ideal situation, we succeeded. We got the job done, on time, though not without overtime. We had the lowest bug-count of any of the teams. And each team member learned a lot from the others, without exception.

6.6. Music

As the team was mostly fairly young, music was a big interest to almost everyone. We added a pair of speakers to one computer, and played a wide variety of music in the team room. My intent was to enable listening to music without the headphones that were prevalent in the cubicles. People brought in CDs, MP3s, and suggested internet radio stations. We cycled through a large range of variety and anyone was allowed to veto anything that was bothering them. For awhile, this worked pretty well. Over time, however, people became dissatisfied. Not only did different

people have different preferences, but preferences of a person varied with their mood. Gradually the headphones started to reappear. I turned off the speakers. Other team members didn't think that the headphones were a big problem. As one person said, it was easy enough to get someone's attention by sending them an "instant message." I was very aware, however, of the reduction in communication on days when headphone use was more prevalent. I'm not sure how best to handle this issue. Fortunately the use of headphones wasn't a constant practice. I would prefer that the team decide not to use them at all, but I don't think I would want to require that.

7. Assessment by team members

I previously mentioned a team member's initial concerns when asked to move into the team room. After the fact, he raved about the experience. "I thought it was great," he told me recently, "I liked it a lot." He talked about the advantages he found working in proximity of other people developing the same component. The way that questions were answered immediately by asking them out loud was a notable plus. The chance to learn, just by osmosis, much more than in other projects was another benefit cited. This squares with the report recently made on the scrumdevelopment yahoogroup by Jeff Heinen [7]

I was having lunch recently with the members of one of our teams. As we discussed their success since adopting Scrum a couple of years ago, they all agreed that "the single most important factor in our success was the team room." This was an interesting comment given that prior to moving into the team room, each and every individual on the team expressed a strong aversion to the idea.

My experience has been that virtually every developer who hasn't experienced a team room is strongly opposed to the idea. It has also been my experience that every developer that tries it ends up having the completely opposite feeling about it.

8. Conclusions

There is more to forming an effective team than just sitting together. People being social animals, will often naturally do whatever else is necessary, though, when working on a common task in close proximity. By enabling informal contact, sitting together enables a team to be highly functional, even in sub-optimal conditions. It's an excellent first step to introducing Agile practices.

A lot of the success in this situation depended on the natural congeniality of the participants. The team members wanted to get along with each other, and wanted to do a good job. I would not recommend leaving this to chance, however. I think the jelling of the team would be more certain and more rapid if the team took an opportunity at regular intervals to notice how things were going. In this case, I think there was a lot of individual introspection. In the future, I would hold frequent retrospectives to involve the team in noticing what was working, and not working. By reflecting on this together, I think the team would jell that much faster and more reliably.

My advice: get out of the cubes. Rub shoulders and elbows. Work together, not just on the same project.

[1] Alexander, Christopher, Sara Ishikawa, and Murray Silverstein, *A Pattern Language*, Oxford University Press, New York, 1977, page 618.

[2] Weinberg, Gerald M., spoken communication at AYE Conference, Phoenix AZ, Nov. 2005.

[3] DeMarco, Tom, and Timothy Lister, *Peopleware, 2nd ed.*, Dorset House Publishing Co., New York, 1999, page 62.

[4] Weinberg, Gerald M., *Quality Software Management, vol 3, Congruent Action*, Dorset House Publishing Co., New York, 1991, p. 224.

[5] Brooks, Frederick P., Jr., *The Mythical Man-Month, Anniversary ed.*, Addison-Wesley Publishing Company, Reading, MA, 1995, page 142.

[6] ibid, page 43.

[7] Heinen, Jeff, "The Scrum/Team Room," Apr 26, 2007, http://groups.yahoo.com/group/scrumdevelopment/message/21231 (accessed Apr 26, 2007).